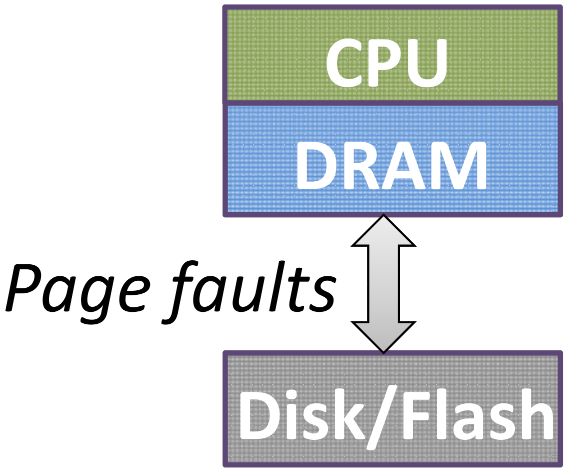
MICRO-47, December 15, 2014

FIRM: Fair and HIgh-PerfoRmance Memory Control for Persistent Memory Systems

Jishen Zhao Onur Mutlu Yuan Xie

New Design Opportunity with NVMs



Main Memory

Load/store

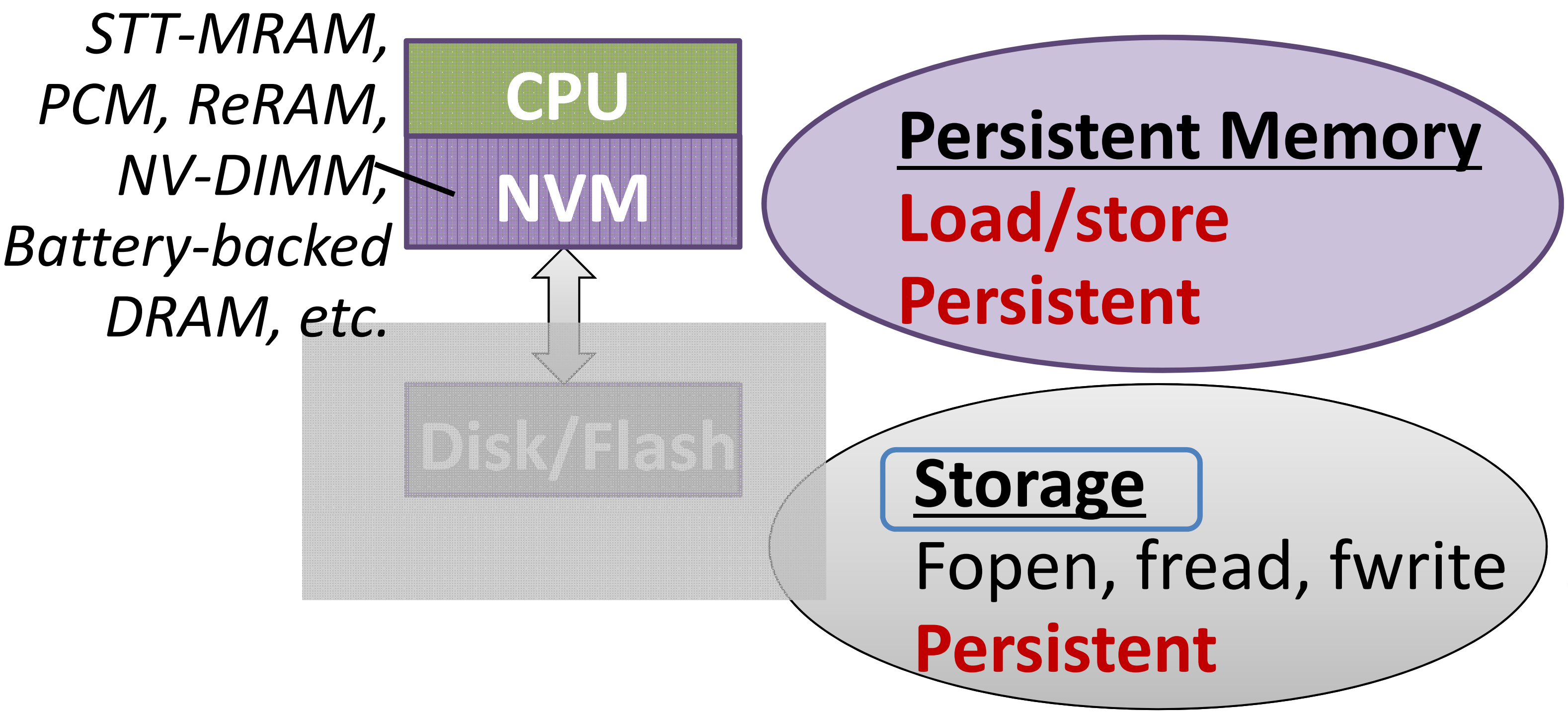
Not persistent

Storage

Fopen, fread, fwrite

Persistent

New Design Opportunity with NVMs



STT-MRAM, PCM, ReRAM, NV-DIMM, Battery-backed DRAM, etc.

CPU

NVM

Disk/Flash

Persistent Memory

Load/store

Persistent

Storage

Fopen, fread, fwrite

Persistent

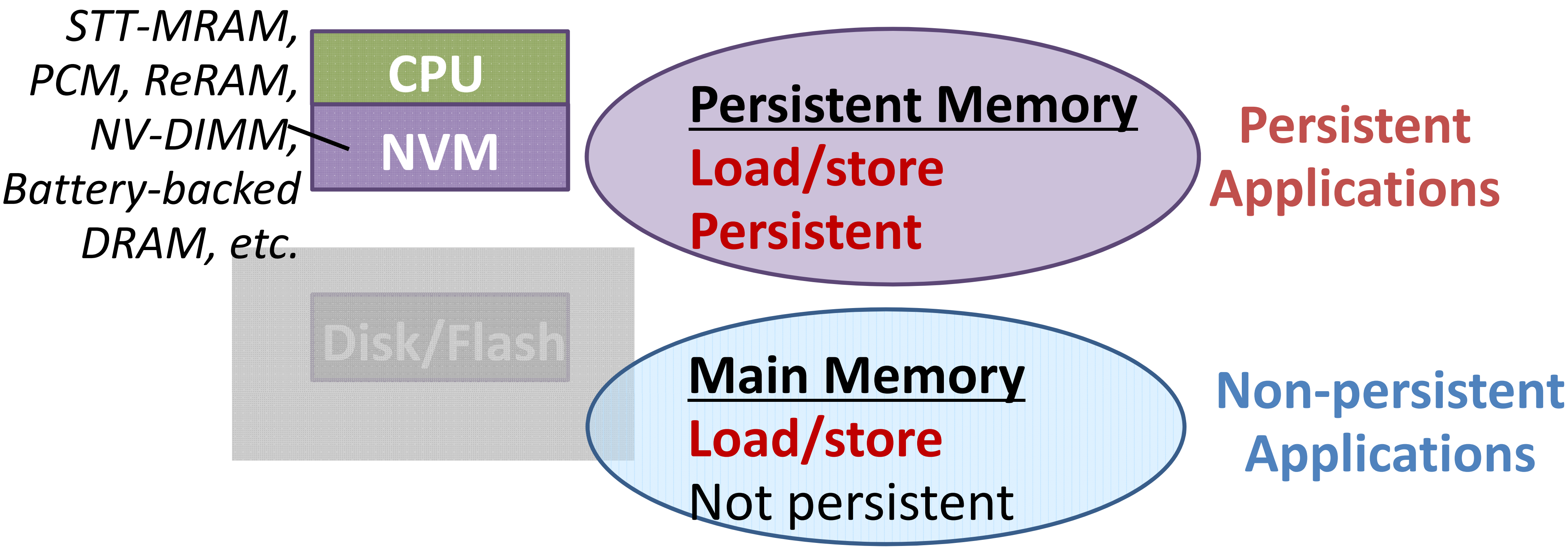
Persistent Applications

Examples applications

Databases, file systems, key-value stores

(In-memory data structures can immediately become permanent)

New Design Opportunity with NVMs



STT-MRAM, PCM, ReRAM, NV-DIMM, Battery-backed DRAM, etc.

CPU

NVM

Disk/Flash

Persistent Memory
Load/store
Persistent

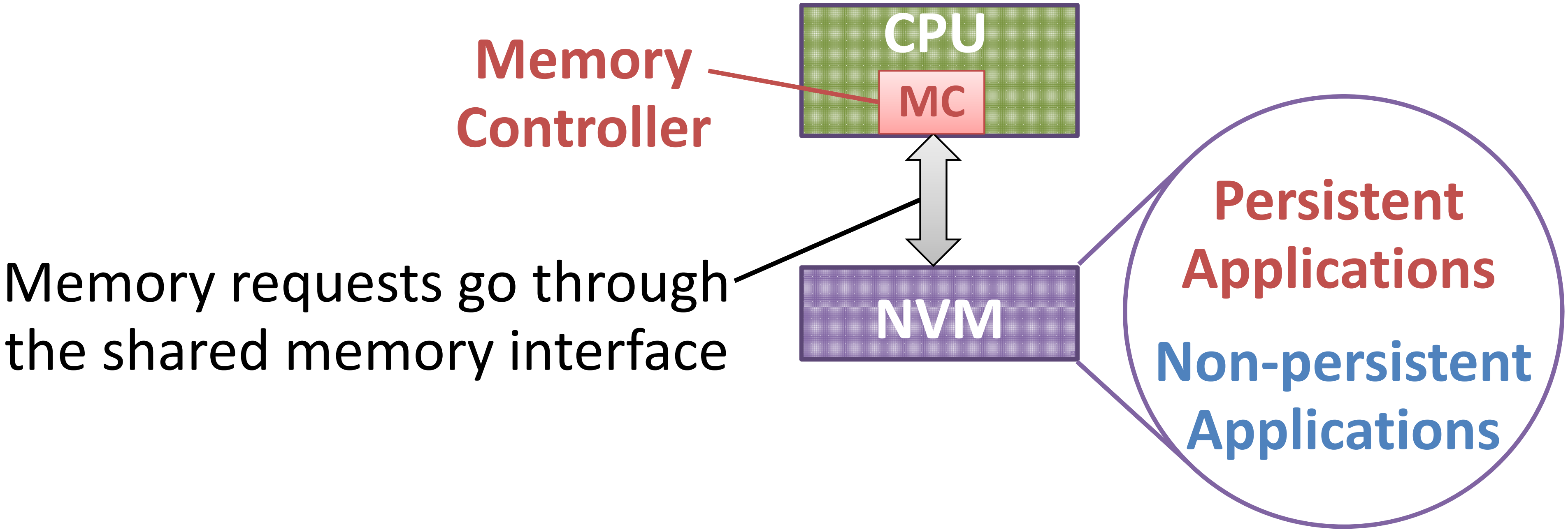
Persistent Applications

Main Memory
Load/store
Not persistent

Non-persistent Applications

New use case of NVM:
concurrently running two types of applications
[Kannan + HPCA'14, Liu + ASPLOS'14, Meza + WEED'14]

Focus of Our Work: Memory Controller Design

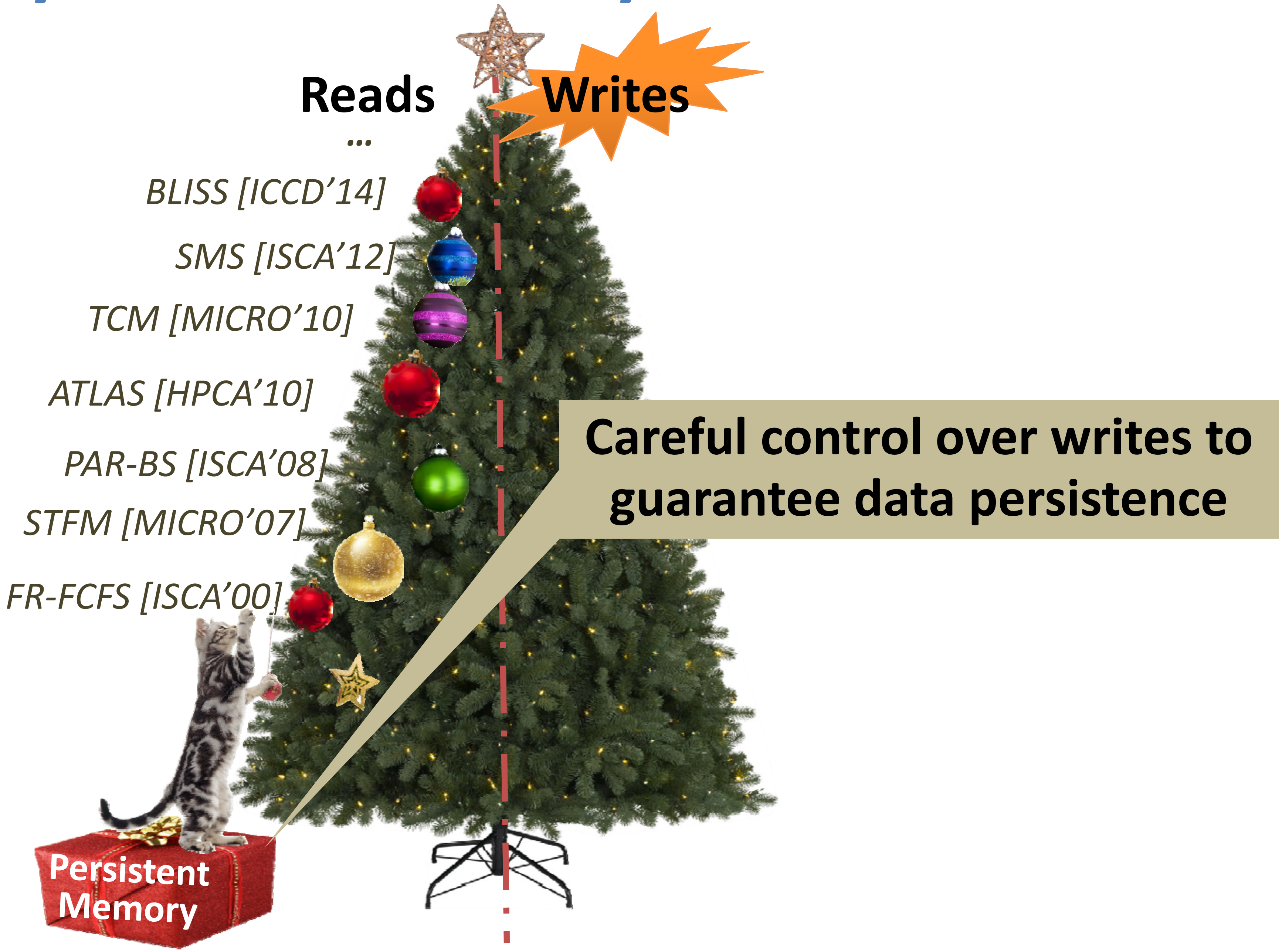


Memory Controller

Memory requests go through the shared memory interface

Fair and High-Performance Memory Control

Why Another Memory Control Scheme?



Reads

…

BLISS [ICCD'14]

SMS [ISCA'12]

TCM [MICRO'10]

ATLAS [HPCA'10]

PAR-BS [ISCA'08]

STFM [MICRO'07]

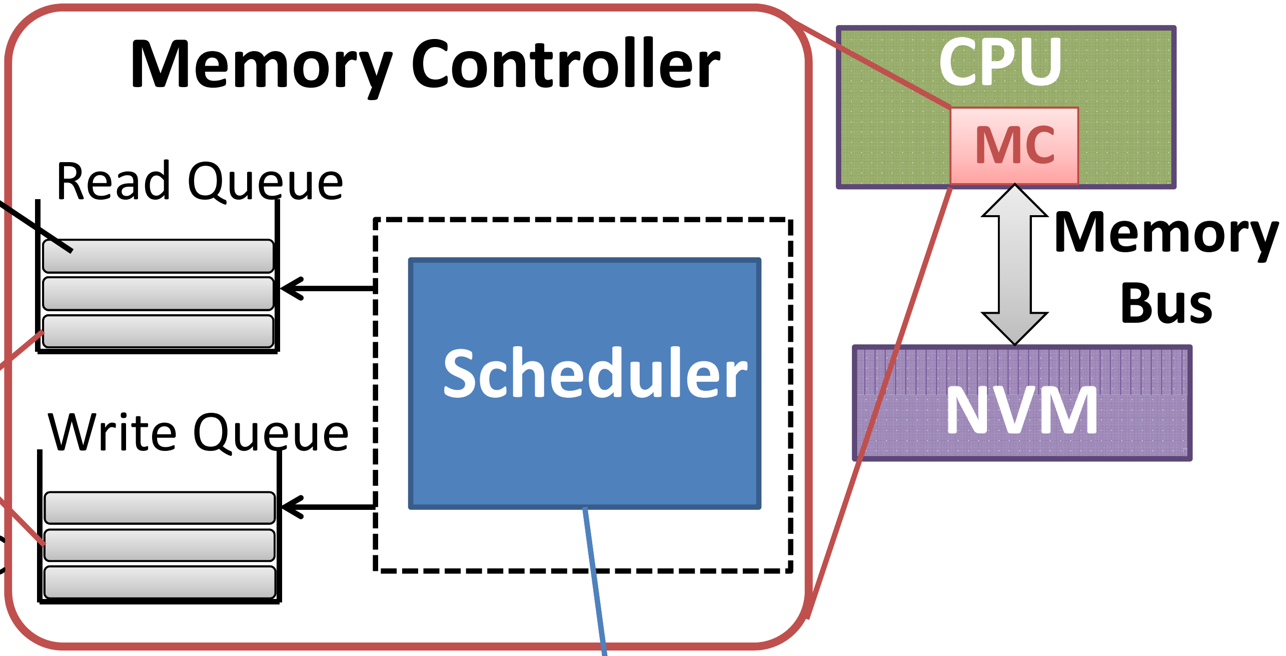
FR-FCFS [ISCA'00]

Writes

Careful control over writes to guarantee data persistence

Persistent Memory

Memory Controller



Memory Requests

Limited number of entries

If one of them overflows, MC needs to **drain** it by stalling the service of the other

Determine which requests can be sent on the memory bus to be serviced

Why conventional memory control schemes are inefficient in persistent memory systems

How to design fair and high-performance memory control in this new scenario

Assumptions and Design Choices

Conventional memory control schemes

Assumptions

1. Reads are on the critical path of application execution

(Application execution is read-dependent)

2. Applications are usually read-intensive

Design choices

1. Prioritize reads over writes

2. Delay writes until they overflow the write queue

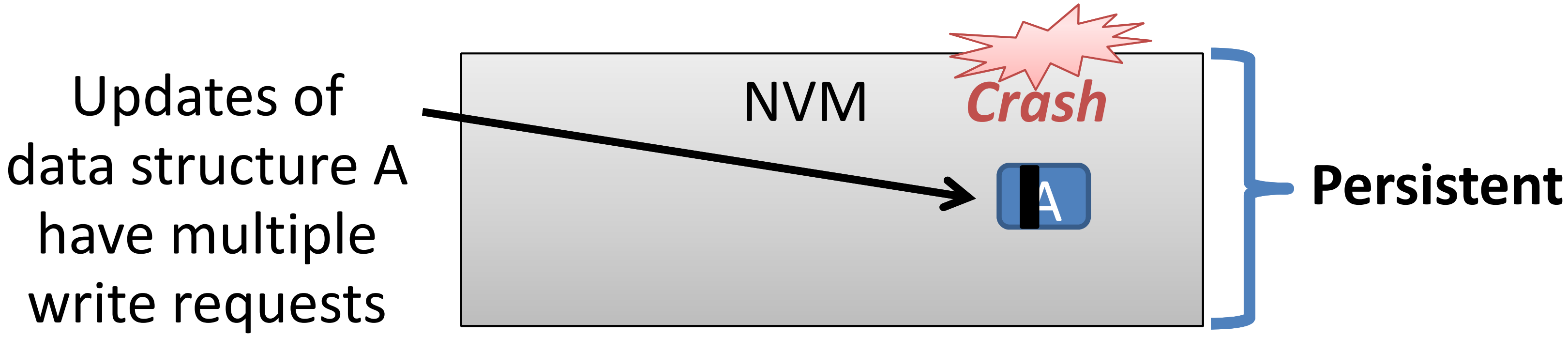
(Infrequent write queue drains)

These assumptions no longer hold in persistent memory, which needs to support data persistence

(Data consistency when the system suddenly crashes or loses power)

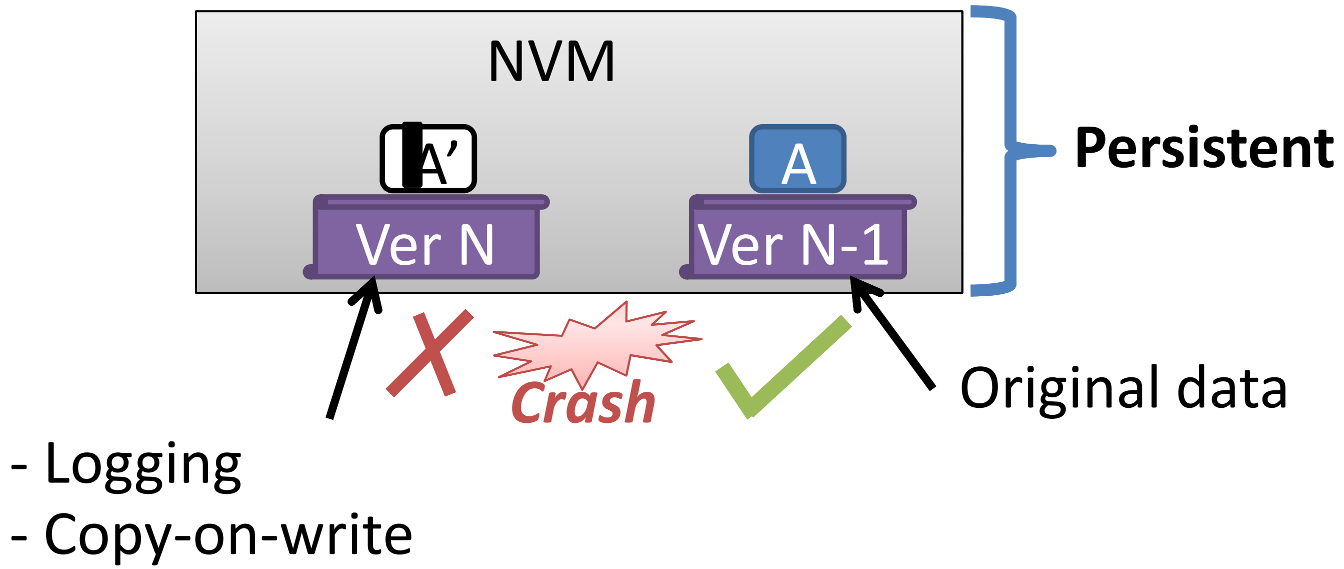
Mechanisms: multiversioning and write-order control

Implication of Multiversioning



Updates of data structure A have multiple write requests

Implication of Multiversioning



NVM

A’

Ver N

A

Ver N-1

Persistent

Crash

Original data

- Logging
- Copy-on-write

The two versions are not updated at the same time

Significantly increasing write traffic –
Two writes with each one data update

[Volos+ ASPLOS’11, Coburn+ ASPLOS’11, Condit+ SOSP’09, Venkataraman+ FAST'11]

Assumptions and Design Choices

Assumptions

1. Reads are on the critical path of application execution

~~2. Applications are usually read-intensive~~

Design decisions

1. Prioritize reads over writes

Persistent applications are usually write-intensive

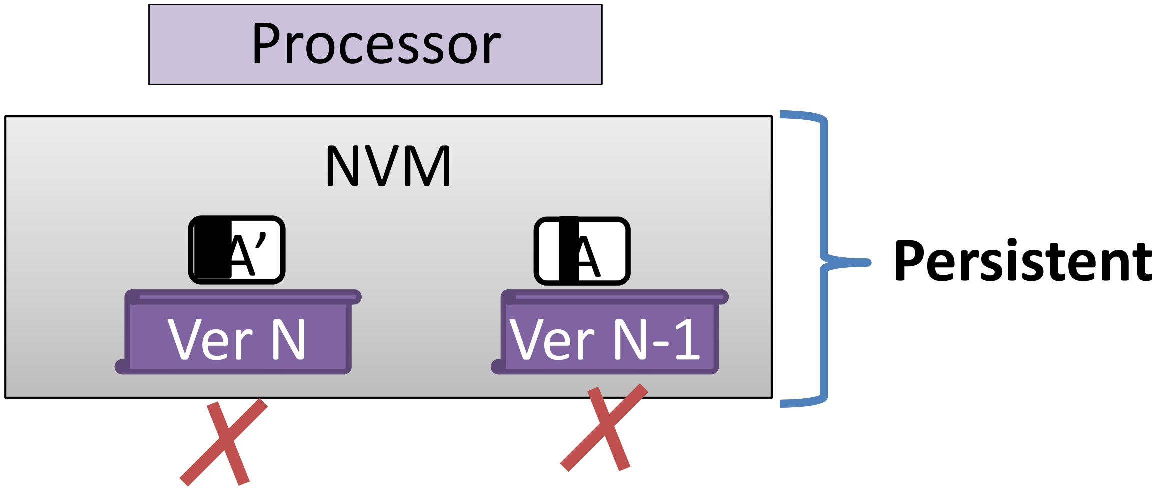
Infrequent write queue drains

Implication of Write-order Control

The two versions are not supposed to be updated at the same time, issued in order:

$A' = \{A'_1, A'_2, A'_3\}$

$A = \{A_1, A_2, A_3\}$



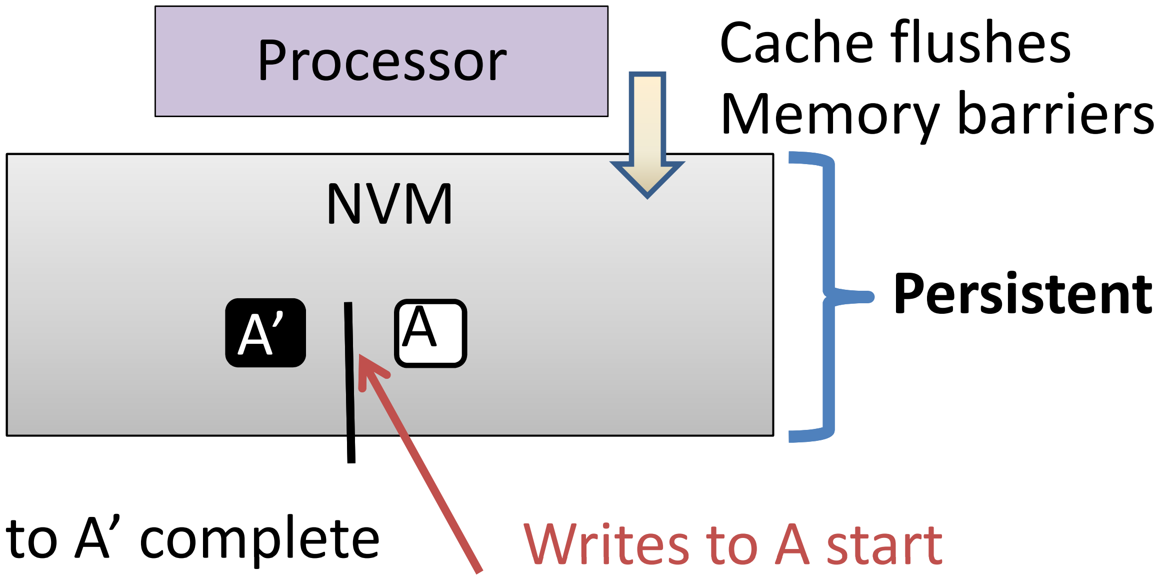
Reordered by caches and MCs:

$A_2, A'_2, A'_1, \mid A_1, A_3$

Crash

Implication of Write-order Control

Issued in order:
Update A’;
Cache flush;
Memory barrier;
Update A ;
Cache flush;
Memory barrier;



Cache flushes
Memory barriers

Persistent

Writes to A’ complete

Writes to A start

Restrict the ordering of writes arriving at the memory

Making application execution write dependent –
Subsequent writes, reads, and computation can all depend on a previously issued write

[Volos+ ASPLOS’11, Coburn+ ASPLOS’11, Condit+ SOSP’09, Venkataraman+ FAST'11]

Assumptions and Design Choices

Assumptions

~~1. Reads are on the critical path of application execution~~

(Application execution is read-dependent)

2 ... e usually read-intensive

Design decisions

1 ... es

Writes are also on the critical execution path

(Application execution is write-dependent)

Persi ... are usually write-intensive

Assumptions and Design Choices

Assumptions

1. Writes are also on the critical execution path

2. Persistent applications are usually write-intensive

Assumptions and Design Choices

Assumptions

1. Writes are also on the critical execution path

2. Persistent applications are usually write-intensive

Design choices

1. Prioritize rea

Unfairness

2. Delay writ
they overflow the w queue

Performance Degradation

Frequent write queue drains

Why ▶

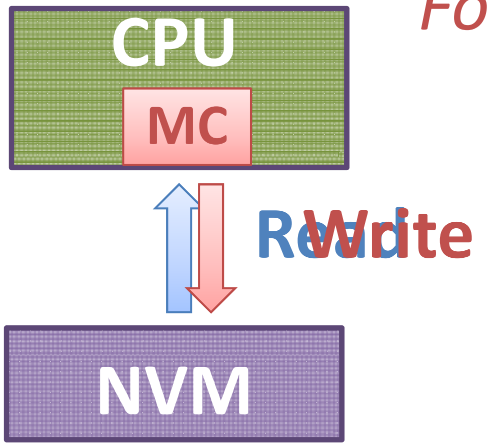
Frequent stall reads to drain the write queue,
frequently switch between
servicing reads and servicing writes

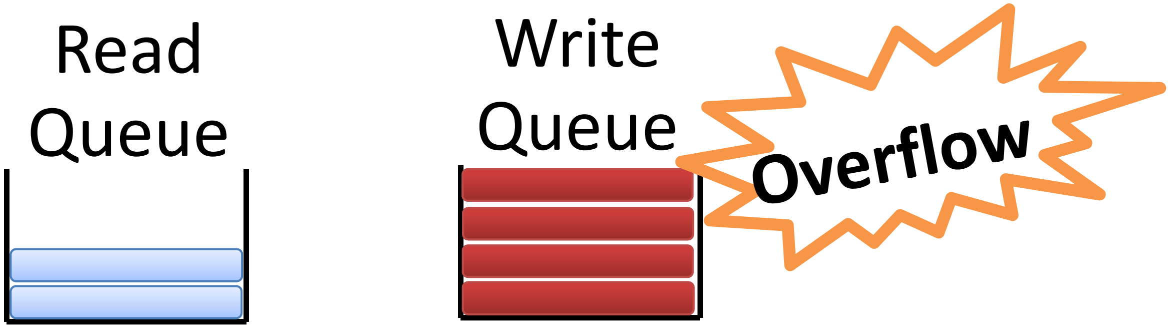
Bus Turnaround Overhead

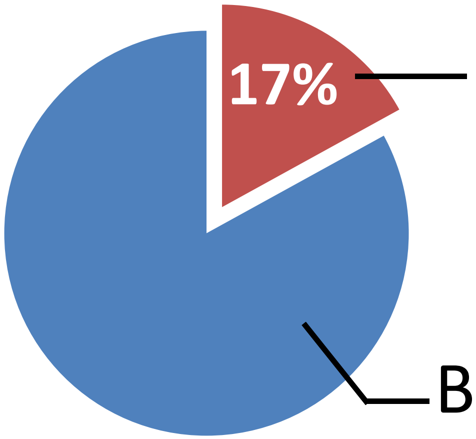
tRTW ~ 7.5ns ***tWTR ~ 15ns***

[Kim + ISCA'12]

Forcing write queue drain by stalling reads







Bus cycles wasted on bus turnarounds (tRTW and tWTR)

Bus cycles to perform memory accesses

Assumptions and Design Choices

Assumptions

1. Writes are also on the critical execution path

2. Persistent applications are usually write-intensive

3. Writes in persistent memory have low bank-level parallelism (BLP)

1. Prioritize rea

2. Delay writ they fill up the wr queue

Unfairness

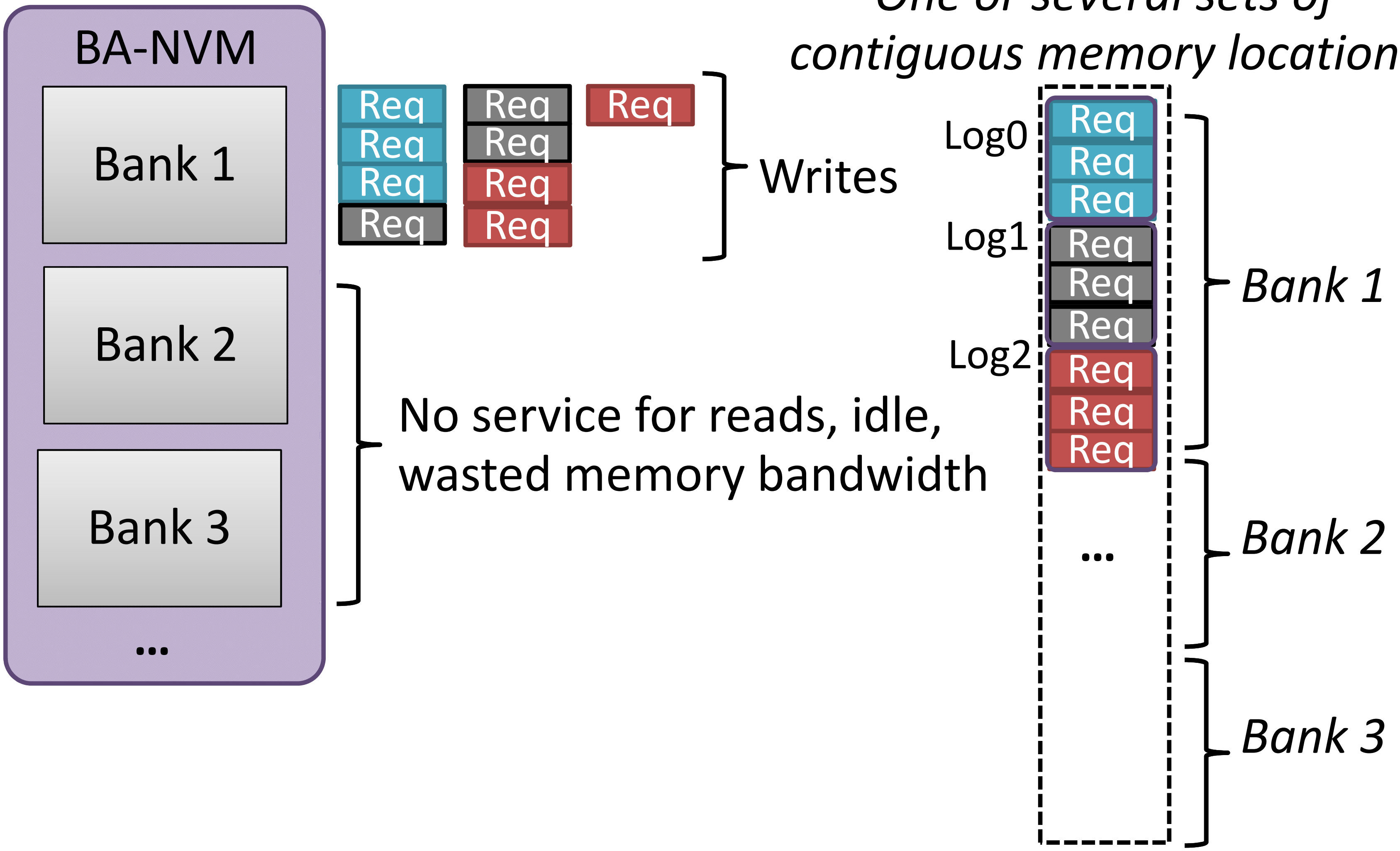
Performance Degradation

Low Bank-level Parallelism (BLP)

Stalling reads for a long time
while the bus is servicing writes to persistent memory

A Log (Spans Multiple Banks)

One or several sets of contiguous memory locations



No service for reads, idle, wasted memory bandwidth

Assumptions and Design Choices

Assumptions

1. Writes are also on the critical execution path

2. Persistent applications are usually write-intensive

3. Writes of persistent applications have low BLP

Design choices

? Fairness

?

High Performance

?

Design Principles

Persistence-Aware Memory Scheduling

Minimizing write queue drains and bus turnarounds, while ensuring fairness

Persistent Write Striding

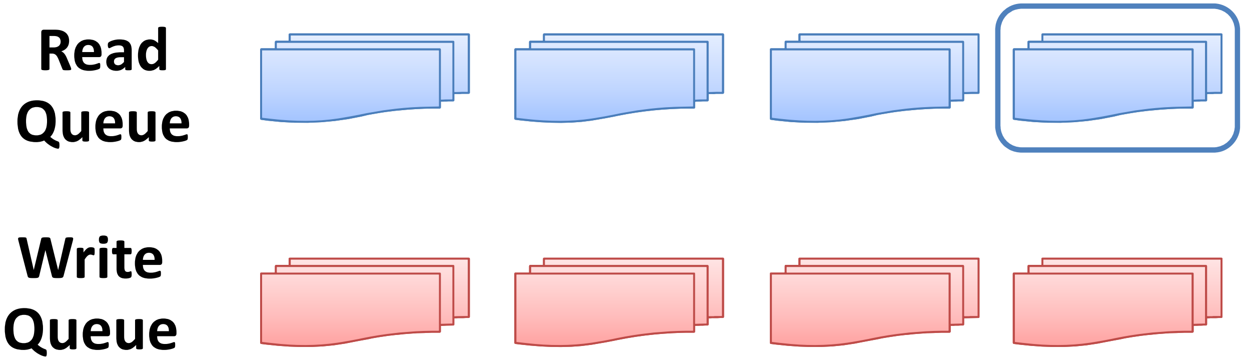
Increasing BLP of writes to persistent memory to fully utilize memory bandwidth

Persistence-aware Memory Scheduling

Minimizing write queue drains and bus turnarounds, while ensuring fairness

Problem: when to switch between servicing read batches and write batches

Low bus turnaround overhead, risk frequent write queue drains



Batch

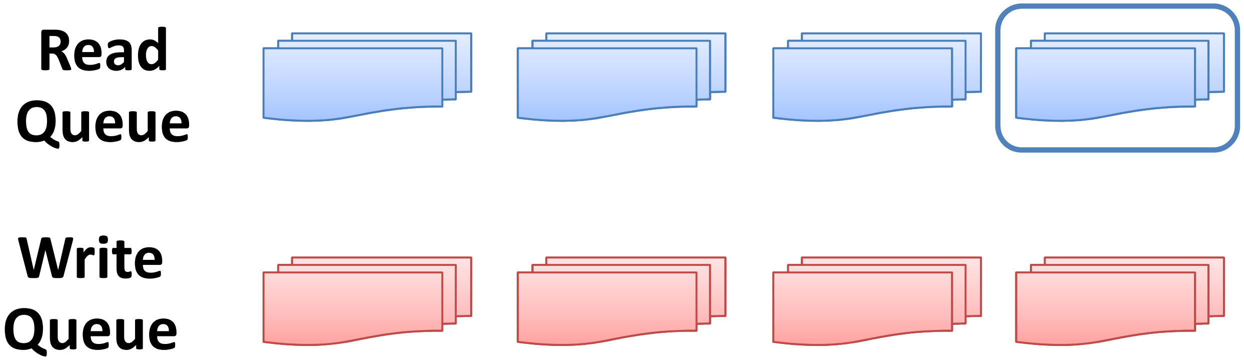
From the same source,
To the same NVM row,
In the same R/W direction

Persistence-aware Memory Scheduling

Minimizing write queue drains and bus turnarounds,
while ensuring fairness

*Problem: when to switch between
servicing read batches and write batches*

Less likely to starve reads and writes, higher bus turnaround overhead



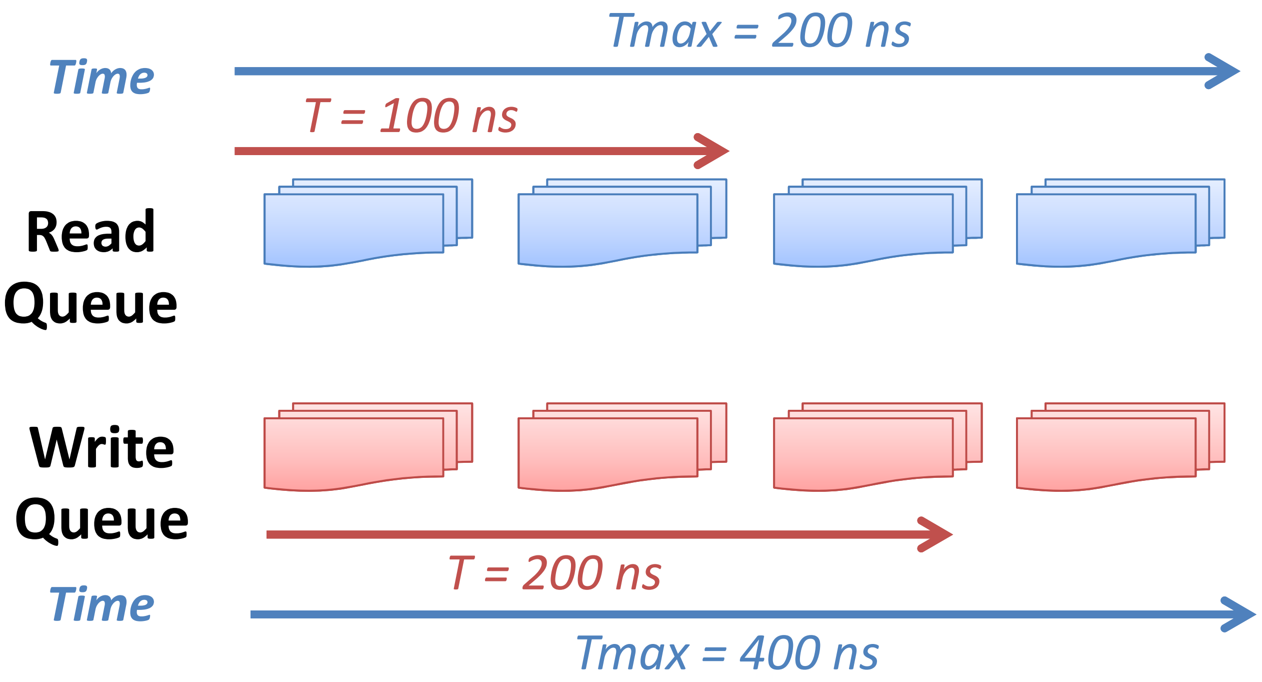
Batch

From the same source,
To the same NVM row,
In the same R/W direction

Persistence-aware Memory Scheduling

Minimizing write queue drains and bus turnarounds, while ensuring fairness

Key idea 1: balance the amount of time spent in continuously servicing reads and writes



Tmax = 200 ns

Time

T = 100 ns

Read Queue

Write Queue

T = 200 ns

Time

Tmax = 400 ns

$$\frac{\textbf{Tmax (read)}}{\textbf{Tmax (write)}} = \frac{1}{2} = \frac{\textbf{T (read)}}{\textbf{T (write)}}$$

Persistence-aware Memory Scheduling

Minimizing write queue drains and bus turnarounds, while ensuring fairness

Key idea 2: Time to service read batches and write batches is JUST long enough

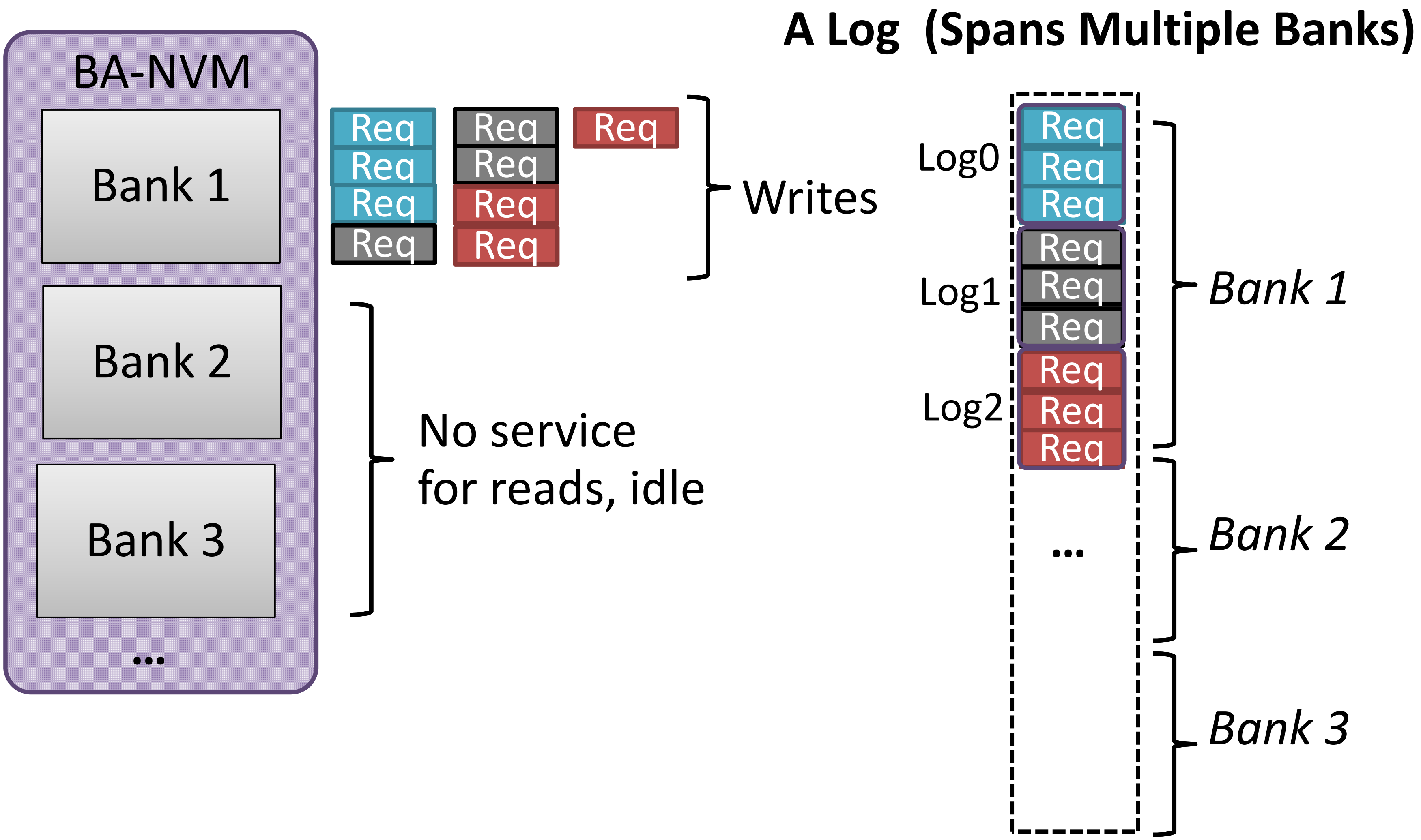
$$\frac{tRTW + tWTR}{T(\text{read batches}) + T(\text{write batches})} < \mu$$

User-defined bus turnaround overhead

Pick the choice with the shortest times

Persistent Write Striding

Increasing BLP of persistent writes to fully utilize memory bandwidth



A Log (Spans Multiple Banks)

BA-NVM

Bank 1

Bank 2

Bank 3

...

Req Req Req Req Req Req Req Req Req

Writes

No service
for reads, idle

Log0 Req Req Req

Log1 Req Req Req

Log2 Req Req Req

...

Bank 1

Bank 2

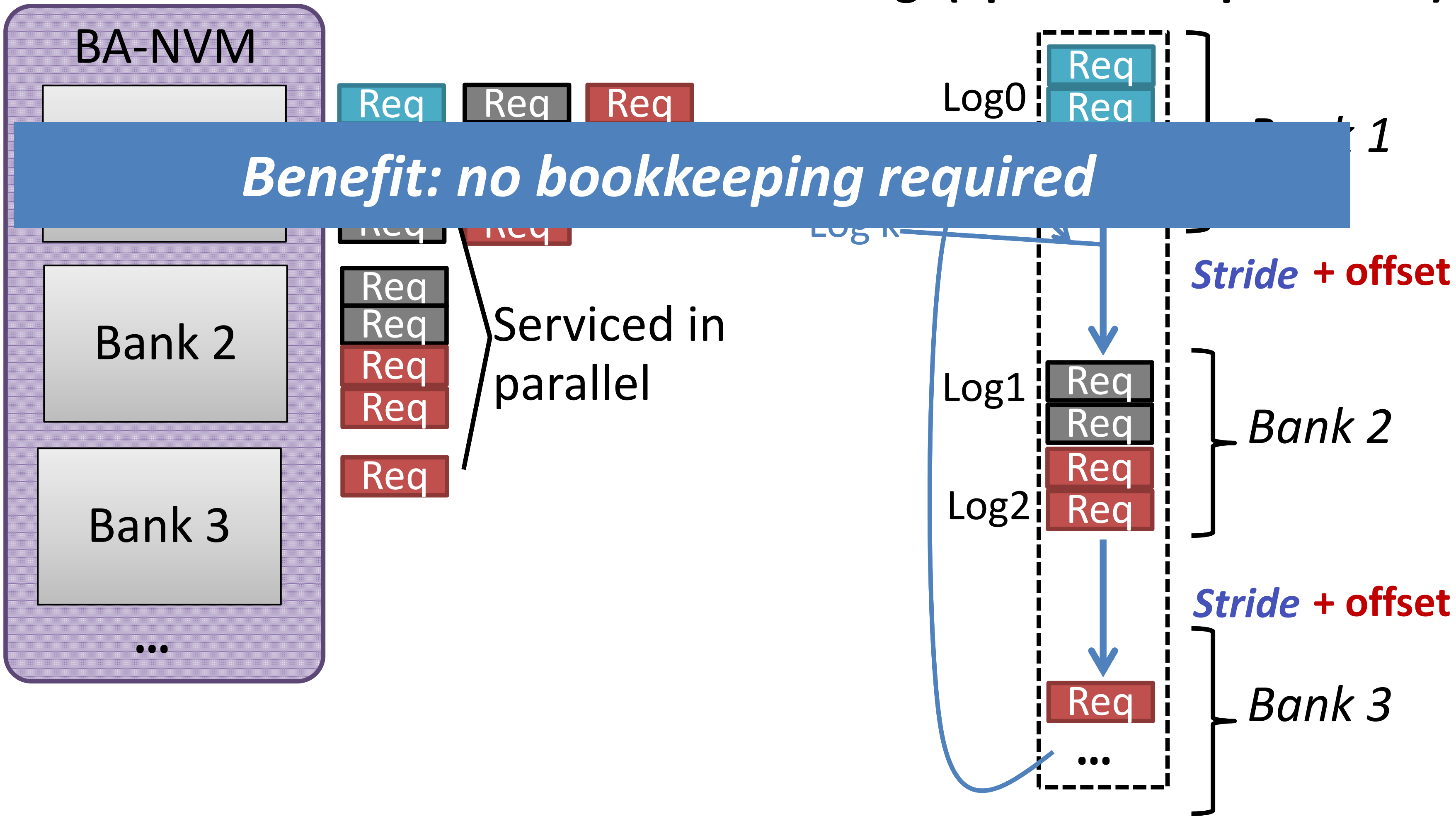
Bank 3

Persistent Write Striding

Increasing BLP of persistent writes to fully utilize memory bandwidth

Key idea: stride writes by an offset to remap them to different banks

A Log (Spans Multiple Banks)



Benefit: no bookkeeping required

Experimental Setup

- Simulator

  - McSimA+ [Ahn+, ISPASS’13] (modified)

- Configuration

  - Four-core processor, eight threads
  - Private caches: L1/L2, SRAM, 64KB/256KB per core
  - Shared last-level cache: L3, SRAM, 2MB per core
  - Main memory: STT-MRAM DIMM (8GB)

- Benchmarks

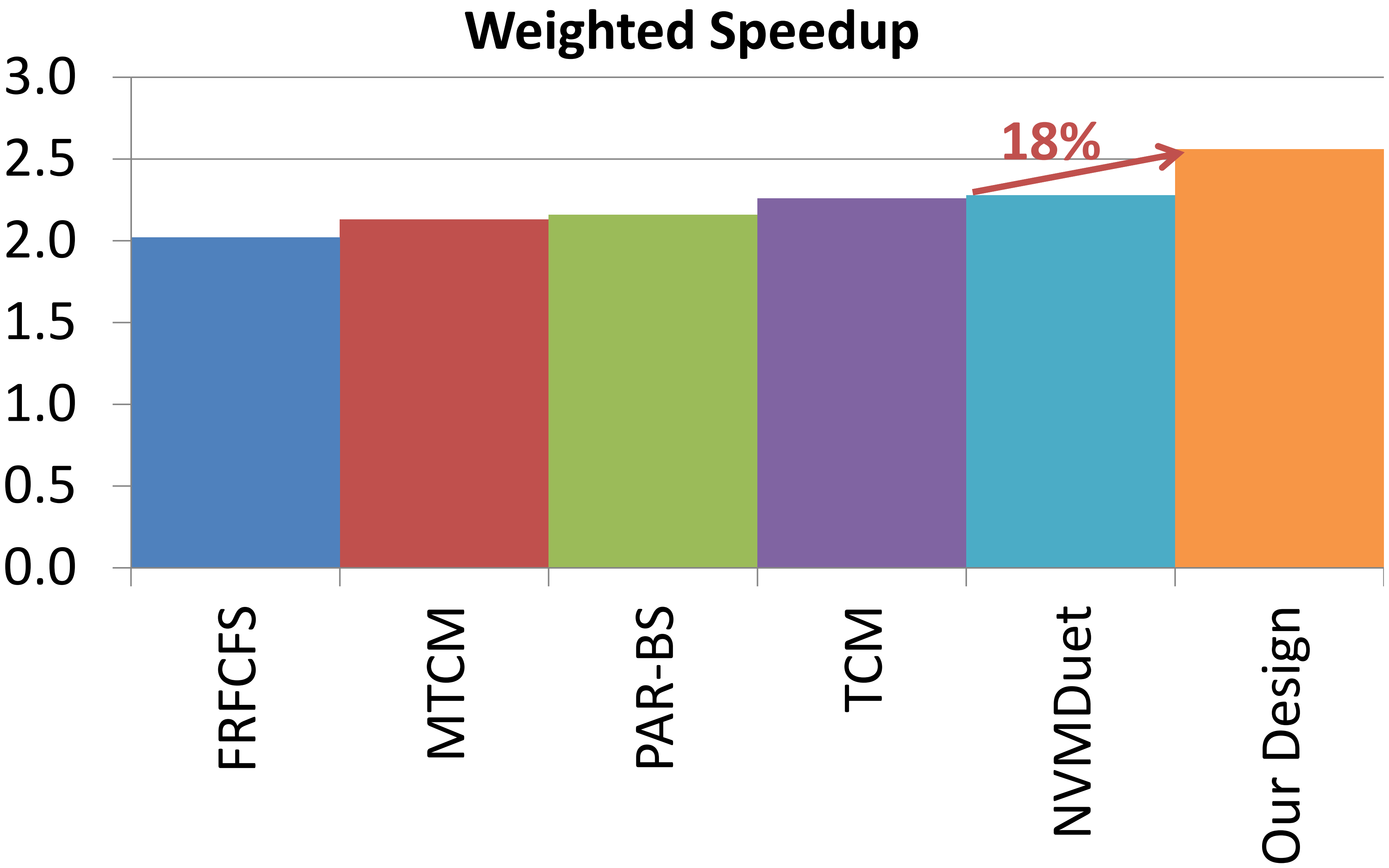
  - 7 non-persistent applications
  - 3 persistent applications

  18 Workloads

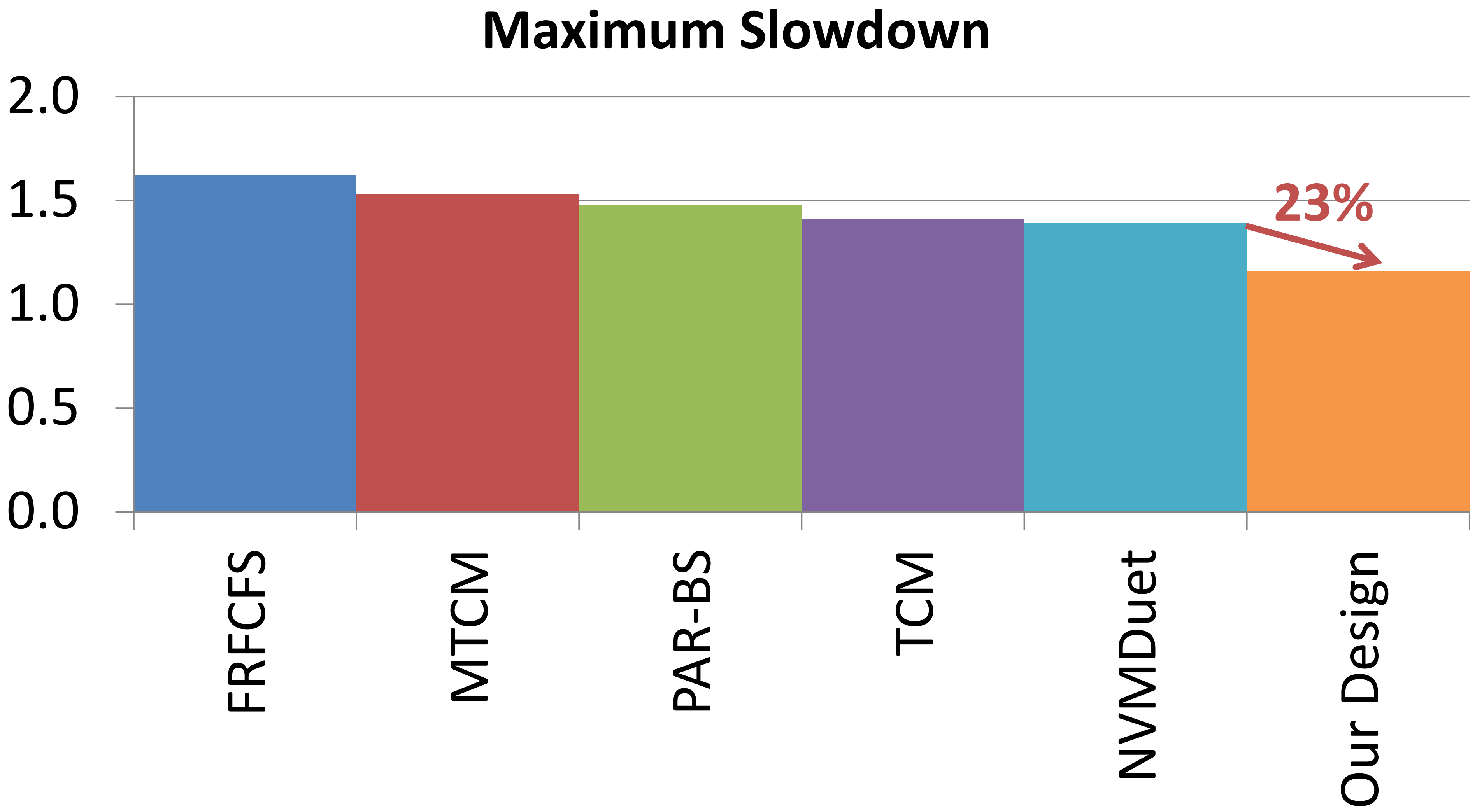
- Metrics

  - Weighted speedup & maximum slowdown

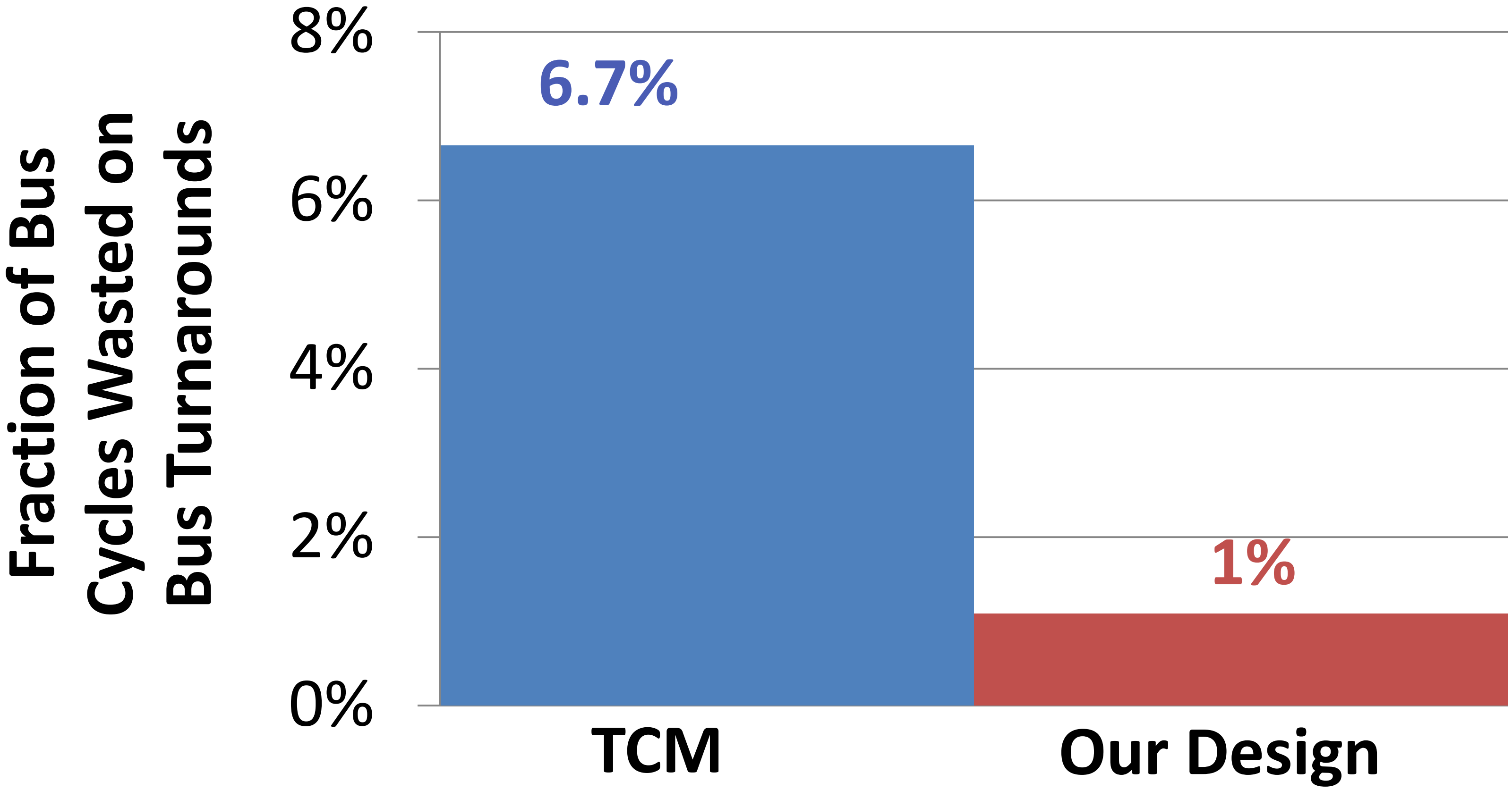
Performance



Fairness


Maximum Slowdown
2.0
1.5
1.0
0.5
0.0
23%
FRFCFS
MTCM
PAR-BS
TCM
NVMDuet
Our Design

Bus Turnaround Overhead



(The worst case of previous designs: 17%)

Other Results

Sensitivity study on various

- NVM Latencies
- Row-buffer sizes
- Number of threads

Summary

Reads **Writes**

...

BLISS [ICCD’14]

SMS [ISCA’12]

TCM [MICRO’10]

ATLAS [HPCA’10]

PAR-BS [ISCA’08]

STFM [MICRO’07]

FR-FCFS [ISCA’00]

Persistent Memory

FIRM [MICRO’14]

Design principles:

Persistence-aware memory scheduling

Persistent write striding

Fairness

High Performance

Thank you!

MICRO-47, December 15, 2014

FIRM: Fair and HIgh-PerfoRmance Memory Control for Persistent Memory Systems

Jishen Zhao Onur Mutlu Yuan Xie

hp

PENNSTATE

1855

SAFARI

Carnegie Mellon

UNIVERSITY OF CALIFORNIA · SANTA BARBARA

LET THERE BE LIGHT